Ontologies in RDF Schema

Lecture 03

1

Ontologies in RDF Schema

- So far, we studied how RDF allows making propositions about individual resources.
- Three key descriptive elements in RDF:
  - **Individuals** (e.g., authors, publishers, recipes).
  - **Relationships** between these individuals.
  - **Types** assignment to literals and resources (e.g., classes like natural numbers or ordered lists).

2

Ontologies in RDF Schema

- Introducing new domains involves defining new terms for:
  - Individuals (e.g., "Nipun", "Chamika")
  - Relations (e.g., "employed by")
  - Types (e.g., "person", "university")
- A collection of these terms for individuals, relations, and classes is called a **vocabulary**.
- Users typically have an intuitive understanding of the vocabulary's meaning.

3

Ontologies in RDF Schema

- A computer system perceives user-introduced terms as plain character strings.
- These terms have no inherent or fixed meaning to the system.
- **Semantic** relationships (i.e., meanings and connections) must be explicitly provided to the system.
- This is necessary for the system to make conclusions based on human-like background knowledge.

4

Ontologies in RDF Schema

- RDF Schema (RDFS) can used in the specification of **background information** (**terminological** or **schema** knowledge) about vocabulary terms.
- RDFS is part of the W3C RDF recommendation.
- It is essentially a **special RDF vocabulary**.
- Every RDFS document is a well-formed RDF document.

5

Ontologies in RDF Schema

- RDFS uses the namespace `http://www.w3.org/2000/01/rdf-schema#`, commonly abbreviated as `rdfs:`.
- RDFS does not provide domain-specific vocabulary.
- Rather it provides **generic language constructs** to define **user-specific vocabularies**.
- These constructs allow **semantic characterization** within the same document.

6

Ontologies in RDF Schema

- An RDFS document can **carry its own semantics**, enabling self-contained definitions.
- This approach avoids the need to **modify software logic** for new vocabularies.
- Any software that supports RDFS can **automatically interpret** RDFS-defined vocabularies **semantically correctly**.

7

Ontologies in RDF Schema

- RDFS can specify schema knowledge, making it a **knowledge representation** or **ontology language**.
- It enables the description of **semantic interdependencies** in a domain of interest.
- The term **"ontology language"** refers to a language that provides a **machine-processable specification** of knowledge.

8

Ontologies in RDF Schema

- In computer science, an **ontology** is a structured description of knowledge with **formally defined meaning**.
- RDFS qualifies as an ontology language because:
  - It provides machine-processable descriptions of domain knowledge.
  - It has a **formal semantics** that defines its meaning.

9

Ontologies in RDF Schema

- Despite its usefulness as an ontology language, it also has its limitations.
- Hence, RDFS is sometimes categorized as a representation language for so-called lightweight ontologies.
- Therefore, more sophisticated applications require more expressive representation languages such as OWL.

10

Classes and Instances

- A key functionality of a knowledge specification formalism is the ability to "type" resources.
- **Typing** means marking resources as elements of a certain **aggregation** or **group**.
- In RDF (Resource Description Framework), this is done using the predicate **`rdf:type`**.
- Generally, the predefined URI `rdf:type` indicates that a resource is an **instance** of a class.

11

Classes and Instances

- In order to clearly separate semantics and syntax:
  - Term “**class**” refers to a set of resources (real-world entities).
  - URIs that represent or refer to these classes are known as **class names**.
- For example, this book can be described as a textbook (which means: a member of the class of all textbooks):

```
book:uri   rdf:type   ex:Textbook .
```


12

Classes and Instances

- It's possible and sometimes reasonable to introduce user-defined class names, depending on the application domain.
- There is no syntactic method to distinguish URIs for individuals (e.g., `book:uri`) from those for classes (e.g., `ex:Textbook`).
- A single URI does not inherently indicate whether it represents an object or a class.
- In some real-world cases, even humans may struggle to determine whether a URI refers to an individual or a class.

13

Classes and Instances

- It is desirable to enforce some clarification by making a definite modeling decision in the context of an RDFS document.
- Therefore, RDFS provides the possibility to indicate class names by explicitly "typing" them as classes.
- For example, it can be specified that, e.g., the class `ex:Textbook` belongs to the class of all classes.

14

Classes and Instances

- This "meta-class" is predefined in the RDFS vocabulary and denoted by the URI `rdfs:Class`.
- Class membership is expressed using `rdf:type`, which characterizes the URI (e.g., `ex:Textbook`) as a class name.

```
ex:Textbook   rdf:type   rdfs:Class .
```


15

Classes and Instances

- On the other hand, the fact that `ex:Textbook` denotes a class.
- This is also an implicit but straightforward consequence of using it as object of a typing statement.
- Hence, the following relationship is derived from the preceding triple.

```
book:uri   rdf:type   ex:Textbook .
```


16

Classes and Instances

- Besides `rdfs:Class`, there are a few further class names predefined in the RDF and RDFS vocabularies and carrying a fixed meaning.
- `rdfs:Resource` denotes the class of all resources.
- `rdf:Property` refers to the class of all properties.
- `rdfs:Literal` represents the class of all literal values.

17

Classes and Instances

- All class names also exhibit a common notational convention:
  - Class names in URIs are typically capitalized.
  - Instance and property names are written in lowercase.
  - Class names can be based on nouns or adjectives (e.g., `ex:Organic` for organic compounds, `ex:Red` for red things).

18

Subclasses

- Suppose an RDFS document contains a single triple: `book:uri rdf:type ex:Textbook.`
- The resource `book:uri` represents the textbook *"Foundations of Semantic Web Technologies"*.
- If a search is performed for instances of the class `ex:Book, book:uri` would **not** be included in the results.
- This is because `ex:Textbook` is not explicitly defined as a subclass of `ex:Book` in the RDFS data.

19

Subclasses

- Human background knowledge allows us to understand that:
  - Every textbook is a type of book.
  - Every instance of the `ex:Textbook` class is also an instance of the `ex:Book` class.
- Automatic systems lack this kind of inherent linguistic background knowledge.
- As a result, such systems cannot infer this logical relationship on their own.

20

Subclasses

- You can explicitly state an additional class membership by adding a triple like:

```
book:uri   rdf:type   ex:Book .
```

- However, this approach causes a recurring problem:
  - Same issue will reappear for every new resource typed as a `textbook` that is added to the RDFS document.

21

Subclasses

- Consequently, for any triple occurring in the document and having the form:

```
u   rdf:type   ex:Textbook .
```

the according triple:

```
u   rdf:type   ex:Book .
```

would have to be explicitly added.

- Moreover, those steps have to be repeated for any new information entered into the document.

22

Subclasses

- This approach increases workload and causes unnecessary verbosity in the specification.
- An alternative would be just to specify that every `textbook` is also a `book`.
- This implies that all textbooks belong to the class of books.
- In ontology terms, `textbook` is a **subclass** of `book`, and `book` is a **superclass** of `textbook`.

23

Subclasses

- RDFS vocabulary provides a predefined way to explicit declaration of subclass relationships using **`rdfs:subClassOf`**.
- For example, following indicate that every textbook is also a book.

```
ex:Textbook   rdfs:subClassOf   ex:Book .
```

- This enables any software that supports the RDFS semantics to identify the individual denoted by `book:uri` as a book even without it being explicitly typed as such.

24

Class Hierarchies

- Subclass statements are commonly used to declare interdependencies between classes.
- They are often used to model entire class hierarchies.
- This is done by specifying the generalization (superclass) to specification (subclass) order.
- E.g.: `Book` is a subclass of `Print Media` and `Print Media` is a superclass of *`Journal`*.

```
ex:Book       rdfs:subClassOf    ex:PrintMedia .
ex:Journal    rdfs:subClassOf    ex:PrintMedia .
```


Class Hierarchies

- RDFS semantics supports the **transitivity** of subclass relationships.
- This means that **"subclasses of subclasses are subclasses."**
- Based on previously stated triples, a new triple can be inferred, even if not explicitly stated.

```
ex:Textbook    rdfs:subClassOf    ex:PrintMedia .
```


Class Hierarchies

- A subclass relationship is **reflexive**: every class is a subclass of itself.
- Reflexivity allows the inference that if `ex:Book` is a class, it is automatically its own subclass.

```
ex:Book    rdfs:subClassOf    ex:Book .
```


Example Class Hierarchies

```
<?xml version="1.0" encoding="utf-8"?> <!DOCTYPE rdf:RDF[
        <!ENTITY ex 'http://example.org/'>
]>

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:ex="http://www.semanticweb-grundlagen.de/Beispiele#">

   <rdfs:Class rdf:about="&ex;Animalia">
       <rdfs:label xml:lang="en">animals</rdfs:label>
   </rdfs:Class>

   <rdfs:Class rdf:about="&ex;Chordata">
       <rdfs:label xml:lang="en">chordates</rdfs:label>
       <rdfs:subClassOf rdfs:resource="&ex;Animalia" />
   </rdfs:Class>

   <rdfs:Class rdf:about="&ex;Mammalia">
       <rdfs:label xml:lang="en">mammals</rdfs:label>
       <rdfs:subClassOf rdfs:resource="&ex;Chordata" />
   </rdfs:Class>
```

```
   <rdfs:Class rdf:about="&ex;Primates">
       <rdfs:label xml:lang="en">primates</rdfs:label>
       <rdfs:subClassOf rdfs:resource="&ex;Mammalia" />
   </rdfs:Class>

   <rdfs:Class rdf:about="&ex;Hominidae">
       <rdfs:label xml:lang="en">great apes</rdfs:label>
       <rdfs:subClassOf rdfs:resource="&ex;Primates" />
   </rdfs:Class>

   <rdfs:Class rdf:about="&ex;Homo">
       <rdfs:label xml:lang="en">humans</rdfs:label>
       <rdfs:subClassOf rdfs:resource="&ex;Hominidae" />
   </rdfs:Class>

   <rdfs:Class rdf:about="&ex;HomoSapiens">
       <rdfs:label xml:lang="en">modern humans</rdfs:label>
       <rdfs:subClassOf rdfs:resource="&ex;Homo" />
   </rdfs:Class>

   <ex:HomoSapiens rdf:about="&ex;SebastianRudolph" />
</rdf:RDF>
```

Class Hierarchies

- Documents with only class hierarchies are known as **taxonomies**.
- Dependencies between subclasses and superclasses are called **taxonomic relations**.
- This modeling approach is intuitive due to its similarity to human conceptual thinking.

29

Properties

- Certain URIs are used in RDF triples as predicates (e.g., `ex:hasIngredient`, `ex:publishedBy`, `rdf:type`).
- These predicate URIs are still URIs and thus denote resources.
- However, interpreting them concretely can be unclear, as terms like "`publishedBy`" don't represent tangible entities.
- Such URIs are not suitable to be considered classes or individuals.

30

Properties

- Instead, they describe relationships between resources or individuals (i.e., the subject and object of an RDF triple).
- Technical term for these predicate URIs is **property**.
- URI `ex:isMarriedTo` represents the set of all married couples.
- Thus, **properties** (like `ex:isMarriedTo`) are more similar to **classes** (sets of things) than to **individuals** (single entities).

31

Properties

- RDF uses the class name **`rdf:Property`** to denote that a URI refers to a property or relation.
- It represents the class of all properties in RDF.
- To state that a specific URI is a property, you use the RDF type statement:

```
ex:publishedBy    rdf:type    rdf:Property .
```


32

Properties

- Note that `rdf:Property` itself denotes a class and not a property. It just contains properties as instances.
- A URI can be identified as a property name either by being explicitly typed or by appearing as the predicate in a triple.
- RDFS semantics ensures that if a triple like `book:uri ex:publishedBy crc:uri` exists, then the triple `ex:publishedBy rdf:type rdf:Property` is a logical consequence.

33

Subproperties

- Properties can be seen as sets of individual pairs, similar to classes.
- This similarity raises the idea of modeling properties like subclass relationships.
- RDFS supports the concept of **subproperties**.
- A subproperty is a more specific version of another property.

34

Subproperties

- Example: `ex:isHappilyMarriedTo` is a subproperty of `ex:isMarriedTo` because happily married couples are a subset of all married couples.
- Above connection can be declared as follows:

```
ex:isHappilyMarriedTo  rdf:subPropertyOf  ex:isMarriedTo.
```


35

Property Hierarchies

- RDFS semantics allows logical inference based on property hierarchies.
- Given the triple:

```
ex:markus   ex:isHappilyMarriedTo   ex:anja .
```

knowing that `ex:isHappilyMarriedTo` is a subproperty of `ex:isMarriedTo`,
it can be inferred that the triple:

```
ex:markus   ex:isMarriedTo   ex:anja .
```


36

Property Hierarchies

- A single subproperty declaration enables automatic recognition of all "happily married" pairs as also "married" in RDFS-compliant systems.
- Note that this way, also properties can be arranged in complex hierarchies, although this is not as commonly done as for classes.

37

Property Restrictions

- Knowing that two entities are connected by a specific property can lead to further conclusions about the entities.
- Such relationships can help infer class or category memberships.
- Example: If one entity is "married to" another, it implies both entities are persons.

38

Property Restrictions

- Predicate `ex:isMarriedTo` implies additional information about its subject and object.
- This additional information can be made explicit using class memberships.
- Specifically, if a triple a

```
a   ex:isMarriedTo   b .
```

- exists, then the following triples should also be asserted:

```
a   rdf:type   ex:Person .
b   rdf:type   ex:Person .
```


39

Property Restrictions

- Adding class membership statements manually in RDF is tedious and repetitive when new data is added.
- A “macro” or “template”-like mechanism is desirable to automate class memberships based on predicates.
- RDFS provides this mechanism using `rdfs:domain` and `rdfs:range`.
  - `rdfs:domain` classifies the subject of a triple.
  - `rdfs:range` classifies the object of a triple.

```
ex:isMarriedTo   rdfs:domain   ex:Person .
ex:isMarriedTo   rdfs:range    ex:Person .
```


40

Property Restrictions

- Literal values can be specified using datatypes (e.g., specifying age as a nonnegative number).

```
ex:hasAge   rdfs:range   xsd:nonNegativeInteger .
```

- Domain and range restrictions serve as the **semantic link** between classes and properties.
- These restrictions are essential for describing the interdependencies between different ontology elements (i.e., classes and properties).

41

An Example

- Suppose that an RDF document contains the following triples:
- This RDFS specification models the existence of "vegetable thai curry",

```
ex:vegetableThaiCurry  ex:thaiDishBasedOn   ex:coconutMilk .
ex:sebastian           rdf:type             ex:AllergicToNuts .
ex:sebastian           ex:eats              ex:vegetableThaiCurry .

ex:AllergicToNuts      rdfs:subClassOf      ex:Pitiable .
ex:thaiDishBasedOn     rdfs:domain          ex:Thai .
ex:thaiDishBasedOn     rdfs:range           ex:Nutty .
ex:thaiDishBasedOn     rdfs:subPropertyOf   ex:hasIngredient .
ex:hasIngredient       rdf:type     rdfs:ContainerMembershipProperty.
```


42

An Example

- It introduces "Sebastian" as a resource who belongs to the class of individuals allergic to nuts.
- It asserts that Sebastian eats the vegetable Thai curry.
- These facts represent **assertional knowledge** (propositions about specific, concrete entities).

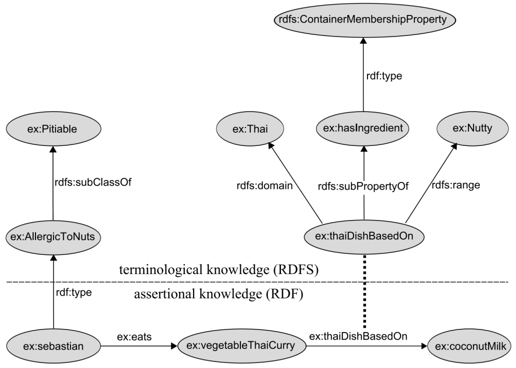
43

An Example

- **Terminological knowledge** (schema-level definitions) includes:
  - Nut-allergic individuals are a subclass of pitiable things.
  - Any Thai dish (based on something) is part of the class of Thai things.
  - Thai dishes are based solely on ingredients from the class of nutty things.
  - If a dish is based on something, it also contains that ingredient.
  - "Having something as ingredient" is defined as a *containedness property*.

44

An Example



45